Matpar: Parallel Extensions for MATLAB

Paul L. Springer
Jet Propulsion Laboratory
4800 Oak Grove Drive, 168-522
Pasadena, CA 91109

pls@volçanoes.jpl.nasa.gov

Telephone: (818) 393-3014
FAX: (818) 393-3134
Presenter: Paul L. Springer
Keywords: parallel, MATLAB, Matpar, client/server

Abstract

Matpar is a set of client/server software that allows a MATLAB user to take advantage of a parallel computer for very large problems. The user can replace calls to certain built-in MATLAB functions with calls to Matpar functions. These Matpar functions are implemented as standard MATLAB external calls (MEX-files) on the client side. The MEX code in turn initiates a session on a parallel computer. The parallel code uses parallel mathematical libraries to produce a solution which is sent back to the calling program, and returned to the user in a seamless fashion.

Introduction

MATLAB is a popular tool among scientists and engineers for matrix and other numerical computations, as well as scientific data visualization. It is a very capable program, and offers a great deal of functionality and flexibility. However, certain problems are so large that they tax the computational resources of even the fastest workstation computers.

For these specific problems it was thought that parallel computation could provide a way to speed up the computation time. This paper describes the software designed for this purpose, called Matpar. Matpar consists of MATLAB extensions known as MEX-files, as well as code that runs on a parallel computer. Also contained in this paper are timings for certain operations to show the improvement in speed obtained by this approach.

Related Work

Efforts to apply parallelism to MATLAB generally fall into two categories: compilers, and interactive routines implemented by means of extensions to the MATLAB language. Because of their ability to parse the entire program, compilers can facilitate task parallelism as well as data parallelism. The tradeoff for this higher degree of parallelism is that the user must go through additional steps, including the compile process, as well as the run on a parallel computer. For users only comfortable with the desktop environment, this can raise a barrier to the use of a compiler. Some examples of parallel compilers for MATLAB include a compiler built at the University of Illinois as part of the Paradigm project[1], the Otter compiler under development at Oregon State University[2], and the CONLAB compiler from the University of Umea in Sweden[3]. RTExpress, from Integrated Sensors Inc., is a development environment which compiles MATLAB script into parallel C code and compiles and links it for the appropriate parallel target platform[4].

Interactive parallelism is generally provided by means of extensions to the MATLAB language. Commands can be typed into MATLAB, the parallel code executed, and the result returned all without any additional steps. The MultiMATLAB project[5], at Cornell University, the PT Parallel Toolbox for MATLAB[6], and the MATLAB toolbox for distributed and parallel computing[7] from the University of Rostock, Germany, all use multiple copies of MATLAB running on the processors of a parallel system, controlled by a master MATLAB process. These three require user input to divide the data between processors. Paramat, a product from Alpha Data Parallel Systems, Ltd., runs on a parallel network of Alpha processors contained on a board which is plugged into a PC host[8]. The user parcels out tasks from the PC running MATLAB to the parallel Paramat software, using special Paramat functions.

Design

An important goal in the design of Matpar was to make the software as easy to use as possible for engineers who may be unfamiliar with parallel computers and who haven't the time to learn a new way of doing things. This distinguishes Matpar from the other methods of applying parallelism to MATLAB: the user does not need to learn how to compile a program, nor does he need to learn how to move his data and program to a parallel computer. The user is also not required to coordinate other copies of MATLAB running on other nodes, and does not need to worry about splitting up the task between the parallel processors.

The Matpar software follows a client/server approach. The client software resides on a workstation, and the server software is on a parallel computer (see Figure 1).

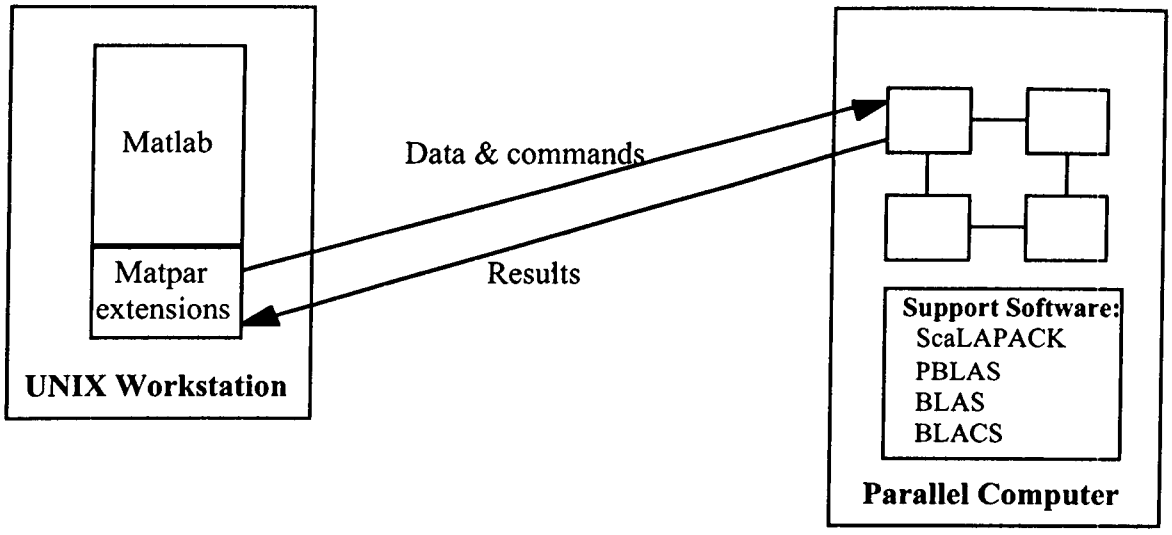

Figure 1: Matpar Architecture

The client software consists of a MATLAB MEX-file for each of the parallel functions, as well as shared object code. The parallel functions have been designed to look as similar to MATLAB commands as possible. For example, the parallel equivalent to MATLAB's qr() function is Matpar's p_qr() function. The MEX-files check the parameters appropriate for the call, and then call a corresponding routine in the shared code. The latter does some additional checking of the parameters, and then uses PVM to initiate a session on the parallel computer. Once the session has begun, the client sends a Matpar request to the parallel computer, again using the PVM communication routines. Each request contains the command to be executed, as well as all the necessary data and parameters for that command. The PVM session is maintained until the user exits MATLAB.

All communications between server and client go through a single node on the parallel computer, called the coordinator node. This decision was made because of the way PVM is implemented on one of the computers to which Matpar has been ported. In the Cray T3D version of PVM, a PVM connection is made only to the first node in the partition being used. In order to make Matpar as portable as possible, it was decided to incorporate this characteristic into the software. This means that the data usually takes two hops, first from the client to the coordinator, and then from the coordinator to the final destination node.

If the data includes a matrix too large to reside on the coordinator node, then the matrix is broken up into blocks. In this case the client determines how to partition

the data among the nodes, and sends each block over separately, along with information about which node is to receive it. The coordinator node forwards it on to the proper destination, in this case using the BLACS[9] communications routines.

Once the data has been received by the parallel computer and properly distributed, the server side of Matpar decides which routines to call. In most cases it calls ScaLAPACK[10] routines. The one exception is in handling Bode plot calculations. In this case the data are replicated on each node, and results generated for each element of a frequency vector. The coordinator node controls this process, directing each computing node to do its calculations for a new frequency value as soon as that node completes its previous calculation. For this operation, the computing node makes calls to LAPACK[11] routines.

After all calculations have been performed on the server side, the server sends the results back to the workstation client. The results are formatted in the way MATLAB expects, so the fact that they were generated by a parallel computer is transparent to the MATLAB user.

Large matrices can result in a communication bottleneck between client and server. For this reason, Matpar provides the option of data persistence for the more expert user. A matrix can be declared persistent by means of the function call p_persist(), which saves that matrix on the parallel computer for future use. Alternatively, a result produced on the server can also be made persistent.

The Matpar software is not a complete parallel version of MATLAB. Instead it is targeted for a subset of MATLAB functions which need a lot of computational power for large matrices. It is then left to the user to decide whether to call one of the parallel routines in place of an equivalent MATLAB function. Because of the overhead involved in sending data from the workstation to the parallel computer, a user would not ordinarily call these routines for small matrices.

Timings

In a memo he wrote concerning a parallelized MATLAB, Cleve Moler, the original designer of MATLAB, stated that one of the reasons they have not developed a parallel version of MATLAB is that data communication times are much longer than computation times[12]. This is true for smaller problems, but for large problems where the computation time is $O(n^3)$, and the communication time is $O(n^2)$, there is a crossover point above which parallelism can be used effectively.

Timings have been done for three hardware configurations In the first configuration, a Sun UltraSparc with 126 MB of memory does all the computation from within

MATLAB itself. In the second configuration the Sun is connected to an Intel Paragon, which has 32 MB on each node, and in the third it is connected to the JPL Cray T3D, with 64 MB per node.

Timings for QR factorization on the three different platforms are shown in Figure 2. The timings on the Sun were for the MATLAB qr() function. The timings for the Paragon and T3D include one-way transmission time for the matrix to be factored. Returning the result took very little time, because for this benchmark, the customer wanted returned only a very small submatrix of the "R" factor.

As can be seen from the figure, the crossover point on the Paragon occurs with a 512 x 512 matrix. The T3D is faster than the Sun for an even smaller size matrix.

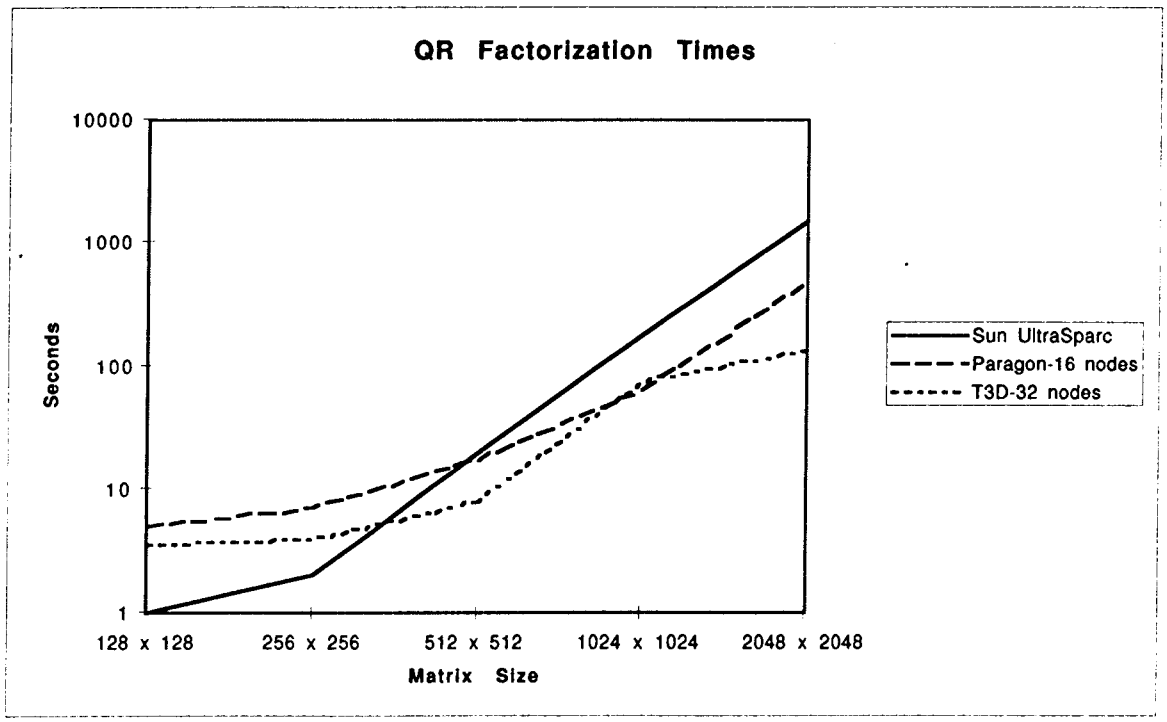

Figure 2: QR Factorization Times

The results of a Bode calculation benchmark are shown in Figure 3, for the same platforms used in the previous test case. The times shown are elapsed times, and include the time for data transmission to and from the parallel platforms. For this case, the crossover point is smaller than for QR factorization: parallelization is effective on both platforms for cases smaller than 256 x 256.

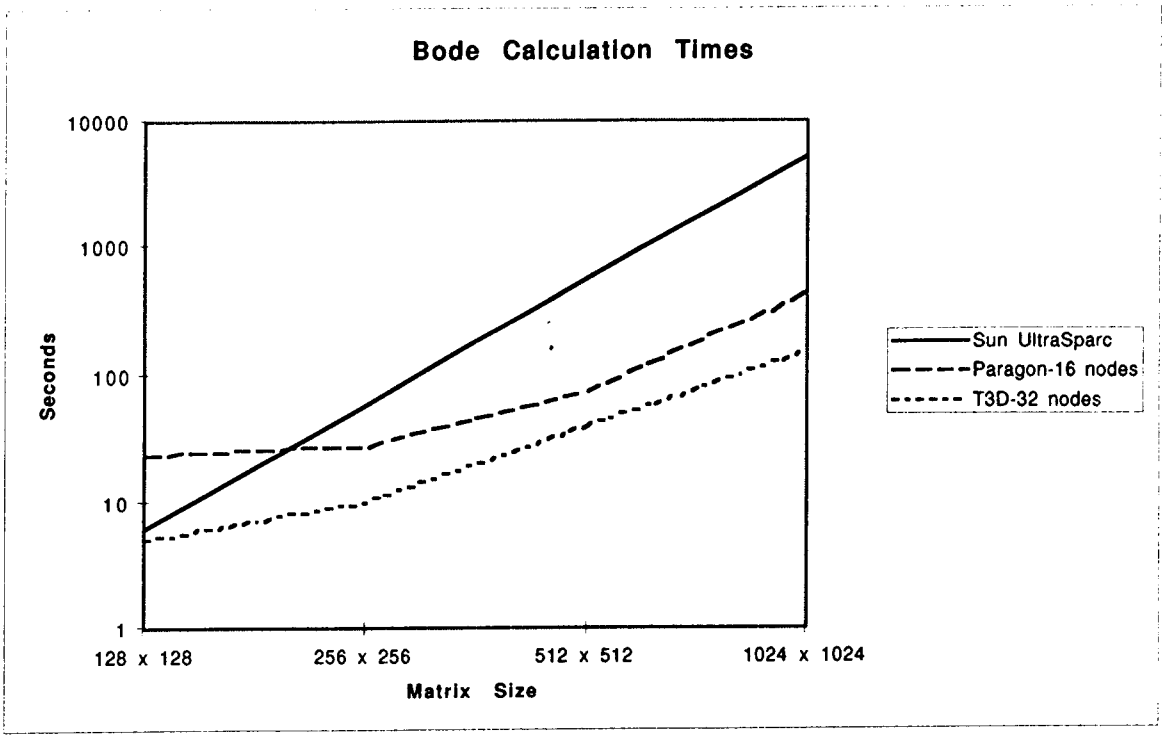

Figure 3: Bode Calculation Times

Conclusion

It is clear from this paper that for computation intensive matrix routines such as QR factorization, matrix-matrix multiplication, Bode plot calculations, and others, the use of a parallel computer offers substantial benefits. In the best case shown here, the 32 nodes of a T3D can do the largest problem over 30 times faster than the Sun. For QR factorization, the crossover point at which the parallel computer and Sun times are equivalent comes with a matrix size of 512 x 512. This crossover point varies for different problems, depending on the amount of computation and communication involved. For problems that are less compute intensive, the crossover matrix size increases. For those problems where persistence can be utilized, communication time decreases, which in turn decreases the crossover matrix size.

Acknowledgments

The work described in this report was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

The T3D Cray Supercomputer used in this investigation was provided by funding from the NASA Offices of Earth Science, Aeronautics, and Space Science.

---

[1] Ramaswamy, S., Hodges, E., Banerjee, P. "Compiling MATLAB Programs to ScsLAPACK: Exploiting Task and Data Parallelism." **Proceedings of IPPS '96**, pp. 613-619.

[2] Quinn, M. "A MATLAB compiler for Parallel Computers". http://www.cs.orst.edu/~quinn/matlab.html

[3] Drakenberg, P., Jacobson, P., and Kågstrom, A. "A CONLAB Compiler for a Distributed Memory Multicomputer." **Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing**, Vol. 2, pp. 814-821, 1993.

[4] "Integrated Sensors Inc." http://www.sensors.com/isi

[5] Trefethen, A., Menon, V., Chang, C., Czajkowski, G., Myers, C., and Trefethen, L. **Multi-MATLAB: MATLAB on Multiple Processors**. Technical Report 96-239, Cornell Theory Center, Ithaca, NY, 1996.

[6] Hollingsworth, J., Liu, K. and Pauca, P. **PT v.1.00 Manual and Reference Pages**, September, 1996.

[7] Pawletta, S., Drewelow, W., Duenow, P., Pawletta, T., and Suesse, M. "A MATLAB Toolbox for Distributed and Parallel Processing." **Proceedings of the MATLAB Conference 95**, Cambridge, MJ, 1995.

[8] Kadlec, J. and Nakhaee, N. "Alpha Bridge: Parallel Processing Under MATLAB." **Proceedings of the Second MathWorks Conference**, 1995.

[9] Dongarra, J., and Whaley, R. **LAPACK Working Note 94: A User's Guide to the BLACS v1.0**. University of Tennessee, Knoxville, TN, 1995.

[10] Choi, J., Dongarra, J., Pozo, R., and Walker, D. "ScaLAPACK: A Scalable Linear Algebra Library for Distributed Memory Concurrent Computers." **Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation**, IEEE Computer Society Press, 1992, 120-127.

[11] Anderson, E., et al. **LAPACK User's Guide**. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992.

[12] Moler, C. "Why there isn't a parallel MATLAB." **MATLAB News & Notes**, Spring, 1995.